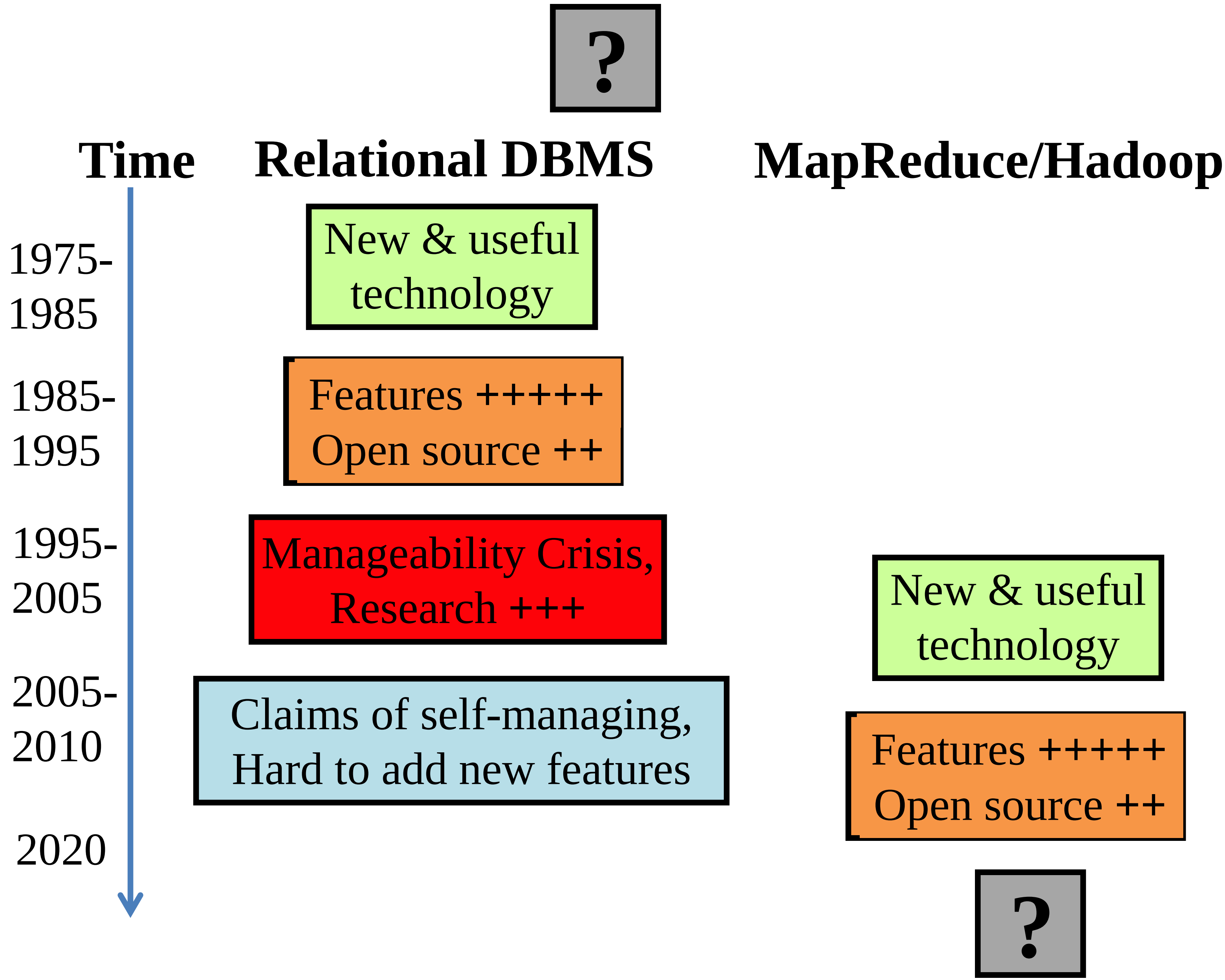
Simplifying Hadoop Usage and Administration

Or, With Great Power Comes Great Responsibility in MapReduce Systems

Shivnath Babu

Duke University


?
Time
Relational DBMS
MapReduce/Hadoop
1975-1985
New & useful technology
1985-1995
Features +++++
Open source ++
1995-2005
Manageability Crisis, Research +++
New & useful technology
2005-2010
Claims of self-managing, Hard to add new features
Features +++++
Open source ++
2020
?

Different Aspects of Manageability

- Testing
- Tuning
- Diagnosis
- Applying fixes
- Configuring
- Benchmarking
- Capacity planning
- Disaster/failure recovery automation
- Detection/repair of data corruption

Roles (often overlap)

- User (writes MapReduce programs, Pig scripts, HiveQL queries, etc.)
- Developer
- Administrator

Lifecycle of a MapReduce Job

File Edit Options Buffers Tools Java Help

```java
public class WordCount {

  public static class Map extends MapReduceBase implements
                Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
  }}}

  public static class Reduce extends MapReduceBase implements
                Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
                   IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) { sum += values.next().get(); }
      output.collect(key, new IntWritable(sum));
  }}

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }}
```

--:--- mapreduce.java All L9 (Java/l Abbrev)--

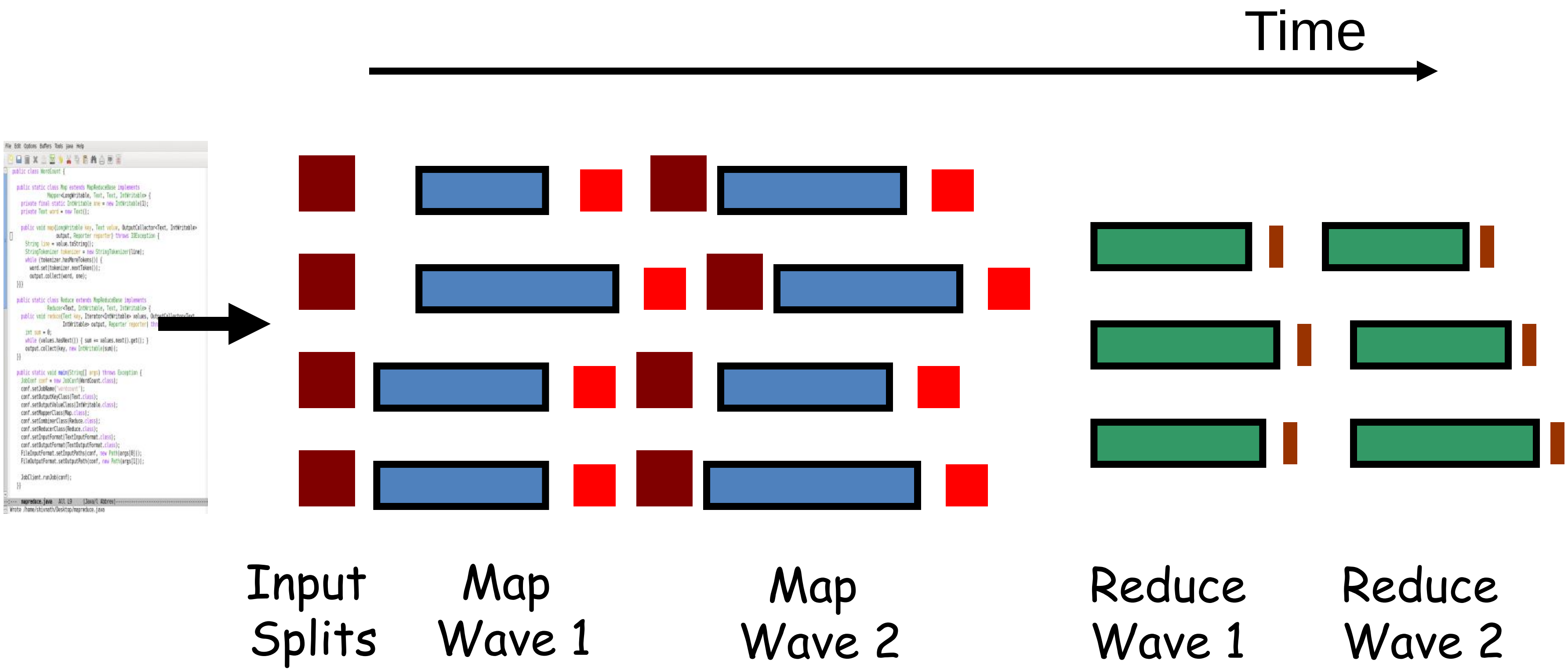
Wrote /home/shivnath/Desktop/mapreduce.java

Map function

Reduce function

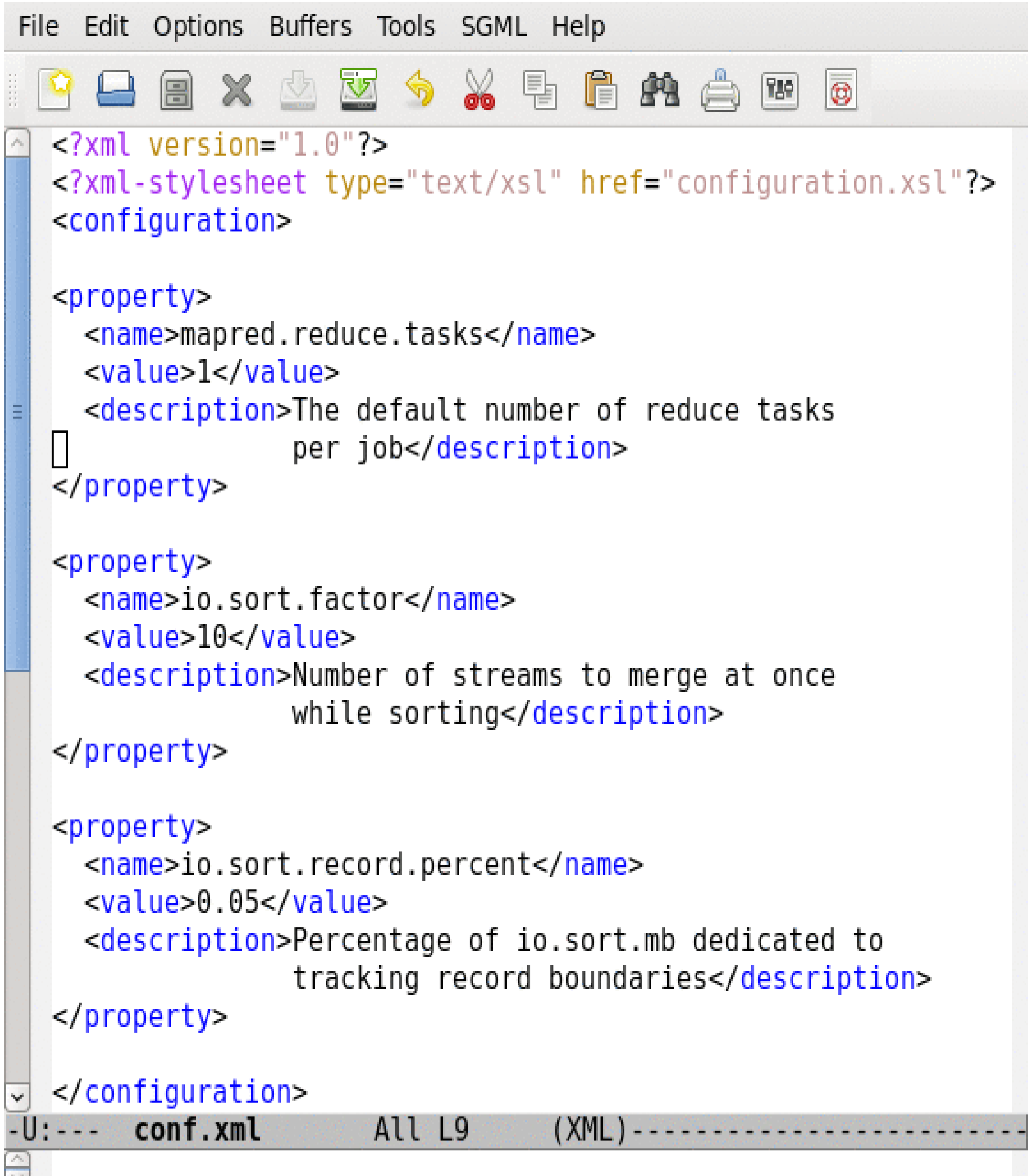
Run this program as a MapReduce job

Lifecycle of a MapReduce Job



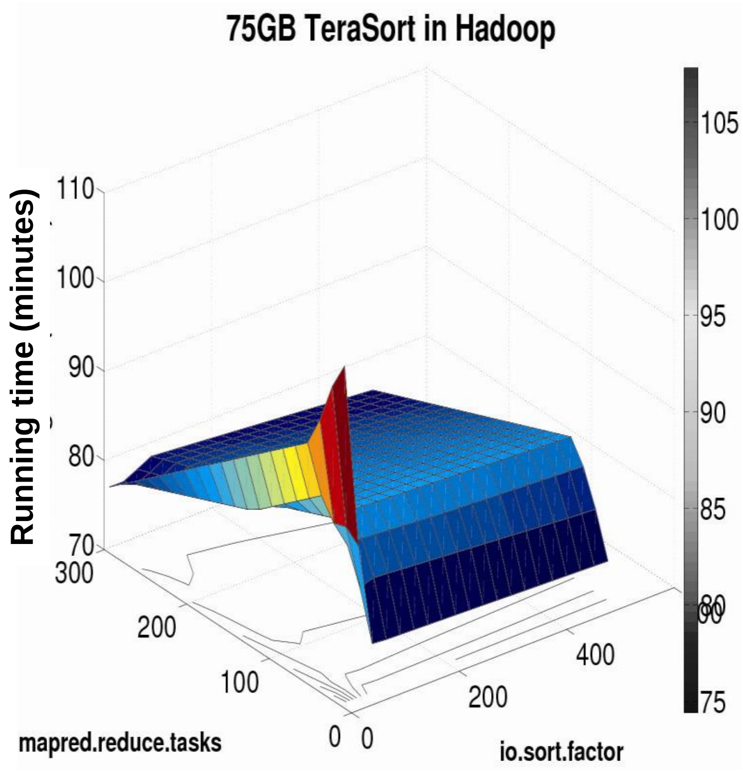
How are the number of splits, number of map and reduce tasks, memory allocation to tasks, etc., determined?

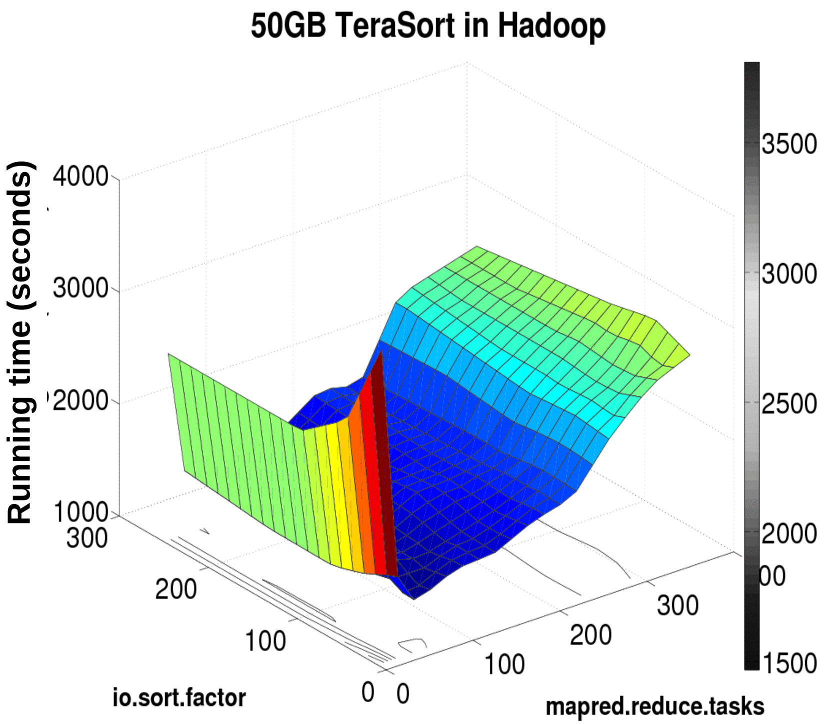
Job Configuration Parameters



- 190+ parameters in Hadoop
- Set manually or defaults are used
- Are defaults or rules-of-thumb good enough?

Experiments

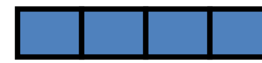




On EC2 and
local clusters

Illustrative Result: 50GB Terasort

17-node cluster, 64+32 concurrent map+reduce slots

mapred.reduce. tasks	io.sort. factor	io.sort.record. percent	Running time
10	10	0.15	
10	500	0.15	
28	10	0.15	
300	10	0.15	
300	500	0.15	

Based on popular rule-of-thumb (→ 28)

- Performance at default and rule-of-thumb settings can be poor
- Cross-parameter interactions are significant

Problem Space

Multi-job workflows

Declarative HiveQL/Pig operations

Job configuration parameters

Complexity

Space of execution choices

Energy considerations

Cost in pay-as-you-go environment

Performance objectives

Current approaches:

- Predominantly manual
- Post-mortem analysis

Is this where we want to be?

Challenges

Features of Hadoop from a usability perspective

- Ability to specify schema late
- Easy integration with programming lang.
- "Pluggability"
  - Input data formats
  - Storage engines
  - Schedulers
  - Instrumentation

These features are very useful when dealing with

- Multiple data formats
- Mix of structured and unstructured data
- Multiple computational engines (e.g., R, DBMS)
- Changes/evolution

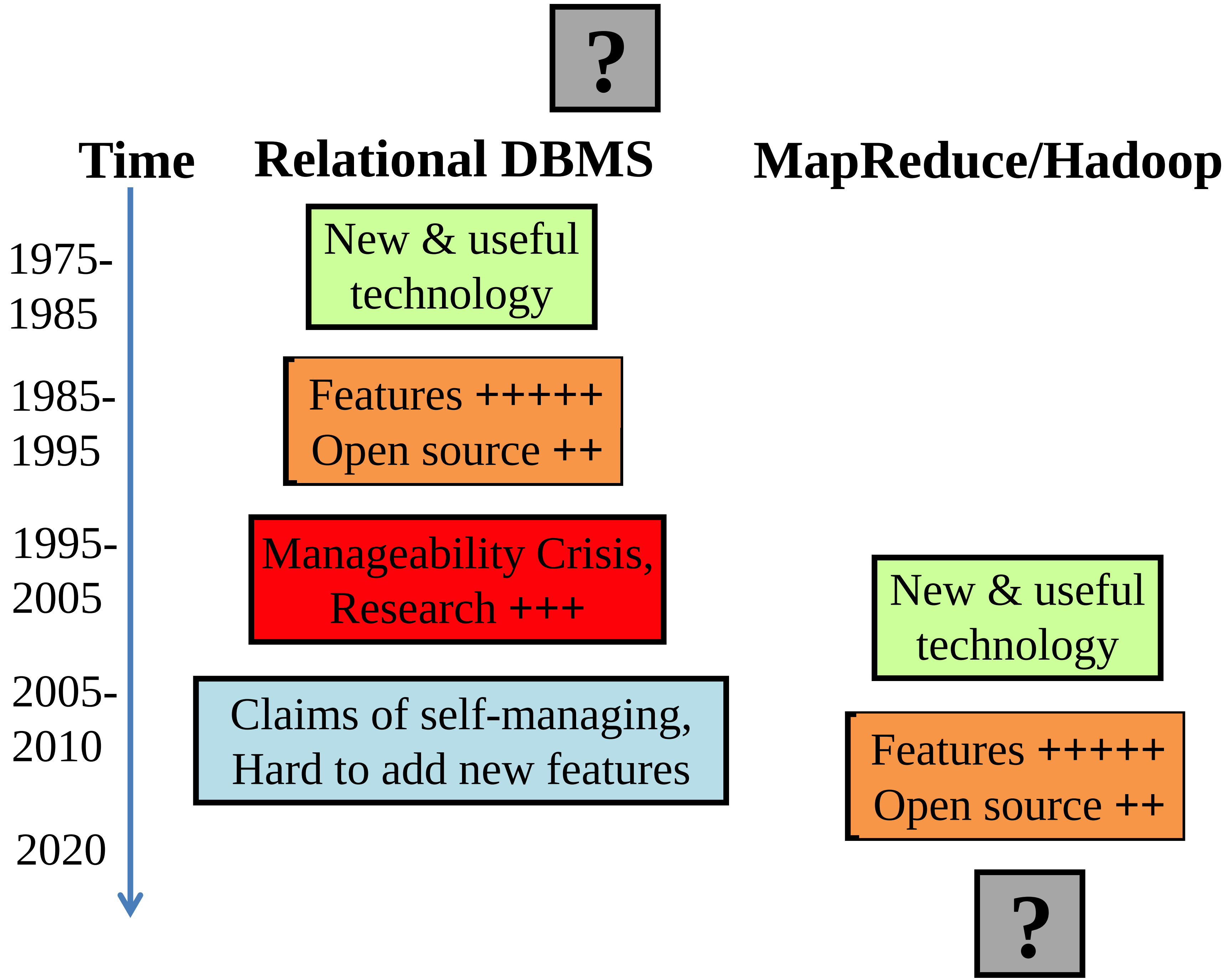
But, they pose nontrivial manageability challenges

Some Thoughts on Possible Solutions

- Exploit opportunities to learn
  - Schema can be learned from Pig Latin scripts, HiveQL queries, MapReduce jobs
  - Profile-driven optimization from the compiler world
  - High ratio of repeated jobs to new jobs is common
- Exploit the MapReduce/Hadoop design
  - Common sort-partition-merge skeleton
  - Design for robustness gives many mechanisms for adaptation & observation (speculative execution, storing intermediate data)
  - Multiple map waves
  - Fine-grained and pluggable scheduler

Some Thoughts on Possible Solutions

- Automate “try-it-out” and “trial-and-error” approaches
  - For example, use 5% of cluster resources to run MapReduce tasks with a different configuration
  - Exploit cloud’s pay-as-you-go resources, EC2 spot instances


?
Time
Relational DBMS
MapReduce/Hadoop
1975-
1985
New & useful
technology
1985-
1995
Features +++++
Open source ++
1995-
2005
Manageability Crisis,
Research +++
New & useful
technology
2005-
2010
Claims of self-managing,
Hard to add new features
Features +++++
Open source ++
2020
?